ULTRIX

digital

Guide to VAX C for ULTRIX

Order Number: AA-ME89B-TE

Guide to VAX C
for ULTRIX

Guide to VAX C for ULTRIX ™

Order Number: AA-ME83B-TE

June 1990

This document describes VAX C constructs in context with both the history of the C programming language and that of the ULTRIX environment on VAX processors. It contains information on VAX C program development in the ULTRIX environment on VAX processor the VAX C programming language, and cross-system portability concerns.

Revision/Update Information: This manual supersedes the *Guide to VAX C for ULTRI* (Order No. AA-KK18A-TE).

Operating System and Version: ULTRIX Version 4.0 or higher

digital equipment corporation
maynard, massachusetts

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may be used or copied only in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by Digital Equipment Corporation or its affiliated companies.

© June 1990.

All Rights Reserved.

The postpaid Reader's Comments forms at the end of this document request your critical evaluation to assist in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

ALL–IN–1
DEC
DEC/CMS
DEC/MMS
DECnet
DECmate
DECsystem–10
DECSYSTEM–20
DECUS
DECwriter
DIBOL
EduSystem
IAS
MASSBUS
PDP
PDT
P/OS
Professional
Q–bus
Rainbow
RSTS
RSX
RT
ULTRIX
UNIBUS
VAX
VAXcluster
VMS
VT
Work Processor

digital™

ZK4585

Contents

Preface xiii

Porting C Programs

Chapter 1 Program Portability Considerations

1.1 Differences Between pcc and VAX C/ULTRIX 1–2
1.1.1 Behavioral Differences 1–2
1.1.1.1 Compiler Phases 1–2
1.1.1.2 Preprocessor Behavioral Differences 1–3
1.1.1.3 Optimization Capabilities 1–3
1.1.1.4 Object File Formats 1–3
1.1.1.5 Listing Output 1–4
1.1.1.6 Default Options 1–4
1.1.1.7 Unavailable Options 1–4
1.1.1.8 Unique Options 1–4
1.1.1.9 Compatibility with lint 1–4
1.1.1.10 Compiler Error Messages 1–5
1.1.2 Language Differences 1–5
1.1.2.1 Function Prototypes 1–6
1.1.2.2 Generic Pointers 1–7
1.1.2.3 The #pragma Preprocessor Directive 1–7
1.1.2.4 Hexadecimal Characters in Escape Sequences 1–7
1.1.2.5 Vacuous Tag Declarations 1–7
1.1.2.6 Additonal Predefined Macros 1–8
1.1.2.7 Storage-Class Specifiers 1–8
1.1.2.8 Storage Class Modifiers 1–8
1.1.2.9 The Variant Structure and Union Declarations 1–8
1.1.2.10 The _align Modifier 1–8
1.1.2.11 The & Operator in Function Calls 1–8
1.1.2.12 Multicharacter Constants 1–8
1.1.2.13 The main_program Option 1–8
1.1.2.14 Specifying Dollar Signs ($) in Identifiers 1–9
1.1.2.15 Order of Evaluation for Subexpressions 1–9
1.1.2.16 Initializing of External Objects 1–9
1.1.2.17 Compiler-Generated Global Symbols 1–9
1.1.2.18 The asm Pseudo Function Call 1–9
1.1.2.19 Variable Initialization 1–9
1.1.2.20 Functions Which Return a Structure Value 1–9
1.1.2.21 Casts as lvalues 1–10

1.2 Differences Between VAX C/VMS and VAX C/ULTRIX 1–10
1.2.1 Language Differences 1–10
1.2.1.1 CDD/Plus 1–10
1.2.1.2 The #include Preprocessor Directive 1–10

1.2.2 Include Files 1–11
1.2.3 Tool Support 1–11
1.2.4 Error Message Formats 1–11
1.2.5 Structure Alignment Differences 1–11
1.2.6 Behavior Differences 1–11
1.2.7 Variable Names 1–12
1.2.8 The #module Preprocessor Directive 1–12

Developing VAX C Programs on ULTRIX

Chapter 2 Developing VAX C Programs for ULTRIX

2.1 ULTRIX Commands for Program Development 2–1

2.2 Creating a VAX C Program 2–3

2.3 Compiling and Linking a VAX C Program 2–3
2.3.1 Functions of the Compiler 2–4
2.3.2 Functions of the Linker 2–4
2.3.3 The vcc Command 2–4
2.3.3.1 Usage Considerations 2–5
2.3.3.2 Specifying Input Files 2–6
2.3.3.3 Specifying Output Files 2–7
2.3.3.4 Options to the vcc Command 2–7
2.3.3.5 Specifying Vendor-Specific Options 2–9

2.4 Compiler and Linker Diagnostic Messages 2–17
2.4.1 Compiler Diagnostic Messages and Error Conditions 2–17
2.4.2 Linker Diagnostic Messages and Error Conditions 2–18

2.5 Compiler Listings 2–18
2.5.1 Source Code Section 2–19
2.5.2 Machine Code Section 2–19
2.5.3 Storage Map Section 2–20

2.6 The lk Linker Image Map 2–23
2.6.1 Object Module Synopsis 2–24
2.6.2 Program Section Synopsis 2–25
2.6.3 Symbol Cross-Reference 2–26
2.6.4 Symbol Value Listing 2–28
2.6.5 Image Synopsis 2–29
2.6.6 Link Run Statistics Synopsis 2–30

Chapter 3 The dbx Debugger

3.1 Invoking the dbx Debugger 3–2

3.2 dbx Conventions 3–3
3.2.1 dbx Initialization Files 3–3
3.2.2 Command Line Retention 3–3
3.2.3 Expressions in dbx Commands 3–3

3.3 Debugging Optimized Programs 3–4

3.4 **dbx Commands** . . . 3–5
3.4.1 Source-Level Debugging Commands . . . 3–7
3.4.2 Machine-Level Debugging Commands . . . 3–13

3.5 **Sample Debugging Session** . . . 3–15

VAX C Programming Concepts

Chapter 4 Program Structure

4.1 **C Programming Language Background** . . . 4–1

4.2 **The VAX C Programming Language** . . . 4–2

4.3 **Writing Your First Program** . . . 4–2

4.4 **Producing Input/Output** . . . 4–4

4.5 **Controlling Program Flow** . . . 4–6
4.5.1 The if Statement . . . 4–7
4.5.2 The switch Statement . . . 4–8
4.5.3 Loops . . . 4–9

4.6 **Values, Addresses, and Pointers** . . . 4–12

4.7 **Aggregates** . . . 4–14
4.7.1 Arrays and Character Strings . . . 4–14
4.7.2 Structures and Unions . . . 4–15

4.8 **Function Definitions** . . . 4–18
4.8.1 The main Function and Function Identifiers . . . 4–20
4.8.2 Parameter List Declarations . . . 4–20
4.8.3 Function Return Data Types . . . 4–21
4.8.4 Variable-Length Parameter Lists . . . 4–22

4.9 **Function Declarations** . . . 4–23

4.10 **Function Prototypes** . . . 4–24

4.11 **Using Parameters and Arguments** . . . 4–26
4.11.1 Function and Array Identifiers as Arguments . . . 4–27
4.11.2 Passing Arguments to the main Function . . . 4–28

4.12 **Identifiers** . . . 4–29

4.13 **Keywords** . . . 4–29

4.14 **Blocks** . . . 4–31

4.15 **Comments** . . . 4–32

4.16 **Source Code Checking Functionality** . . . 4–32

Chapter 5 Statements

5.1 **Control Flow Statements** 5–1
5.1.1 The null Statement 5–1
5.1.2 The goto Statement 5–2
5.1.3 The label Statement 5–2

5.2 **Expressions and Blocks as Statements** 5–2
5.2.1 The expression Statement 5–2
5.2.2 The compound Statement 5–2

5.3 **Conditional Statements** 5–3
5.3.1 The if Statement 5–3
5.3.2 The switch Statement 5–4

5.4 **Looping Statements** 5–6
5.4.1 The for Statement 5–6
5.4.2 The while Statement 5–7
5.4.3 The do Statement 5–7

5.5 **Interrupting Statements** 5–8
5.5.1 The break Statement 5–8
5.5.2 The continue Statement 5–8
5.5.3 The return Statement 5–9

Chapter 6 Expressions and Operators

6.1 **The lvalues and rvalues** 6–1

6.2 **Primary Expressions and Operators** 6–2
6.2.1 Parenthetical Expressions 6–2
6.2.2 Function Calls 6–2
6.2.3 Array References ([ ]) 6–3
6.2.4 Structure and Union References 6–4

6.3 **Overview of the VAX C Operators** 6–4

6.4 **Unary Expressions and Operators** 6–7
6.4.1 Negating Arithmetic and Logical Expressions 6–7
6.4.2 Incrementing and Decrementing Variables 6–7
6.4.3 Computing Addresses and Dereferencing Pointers 6–8
6.4.4 Calculating a One's Complement 6–9
6.4.5 Forcing Conversions to a Specific Type Using the Cast Operator 6–9
6.4.6 Calculating Sizes of Variables and Data Types (sizeof) 6–10

6.5 **Binary Expressions and Operators** 6–10
6.5.1 Additive Operators 6–10
6.5.2 Multiplication Operators 6–11
6.5.3 Equality Operators 6–11
6.5.4 Relational Operators 6–11
6.5.5 Bitwise Operators 6–12
6.5.6 Logical Operators 6–13
6.5.7 Shift Operators 6–13

6.6 **The Conditional Expression and Operator** 6–13

6.7 **Assignment Expressions and Operators** 6–14

6.8 **The Comma Expression and Operator** 6–15

6.9 **Data-Type Conversions** 6–15
6.9.1 Converting Operands 6–16
6.9.2 Converting Function Arguments 6–17

Chapter 7 Data Types and Declarations

7.1 **Constants** 7–1

7.2 **Variables** 7–1
7.2.1 Data-Type Keywords 7–2
7.2.2 Format of a Variable Declaration 7–2

7.3 **Integers (int, long, short, char, and unsigned)** 7–3
7.3.1 Integer Constants 7–4
7.3.2 Character Constants 7–5
7.3.3 Escape Sequences 7–5

7.4 **Floating-Point Numbers (float and double)** 7–6

7.5 **Floating-Point Constants** 7–7

7.6 **Pointers (*)** 7–8
7.6.1 **void** Pointers 7–9

7.7 **Enumerated Types** 7–9

7.8 **Arrays ([ ])** 7–11

7.9 **Initializing Arrays** 7–13

7.10 **Character-String Variables (char * and char [ ])** 7–14

7.11 **Character-String Constants** 7–14

7.12 **Structures and Unions (struct and union)** 7–15
7.12.1 Declaring a Structure or Union 7–16
7.12.2 Referencing Members of Structures or Unions 7–17
7.12.3 Initializing Structures 7–18
7.12.4 Variant Structures and Unions 7–20
7.12.5 Bit Fields 7–22

7.13 **The void Keyword** 7–22

7.14 **The typedef Keyword** 7–23

7.15 **Interpreting Declarations** 7–23

Chapter 8 Storage Classes and Allocation

8.1 Scope 8–1
8.1.1 The Compilation and Linking Process 8–2
8.1.2 Position of the Declaration 8–2
8.1.3 Lexical Scope and Link-Time Scope 8–3
8.1.4 Program Example 8–4

8.2 Storage Allocation 8–7

8.3 Internal Storage Class 8–7
8.3.1 The auto Specifier 8–8
8.3.2 The register Specifier 8–9

8.4 The Static Storage Class 8–9

8.5 The External Storage Class 8–10

8.6 The Global Storage Class 8–11
8.6.1 The globaldef and globalref Specifiers 8–11
8.6.2 Comparing the Global and the External Storage Classes 8–13
8.6.3 The globalvalue Specifier 8–14
8.6.4 Global Enumerated Types 8–15

8.7 Data-type modifiers 8–16
8.7.1 The const Modifier 8–16
8.7.2 The volatile Modifier 8–17

8.8 Storage-Class Modifiers 8–17
8.8.1 The noshare Modifier 8–18
8.8.2 The readonly Modifier 8–18
8.8.3 The _align Modifier 8–18

Chapter 9 Preprocessor Directives

9.1 Macro Definitions (#define and #undef) 9–1
9.1.1 Constant Identifiers 9–3
9.1.2 Macro Parameters 9–3
9.1.3 Listing Substituted Lines 9–5
9.1.4 Canceling Definitions (#undef) 9–5

9.2 Conditional Compilation (#if, #ifdef, #ifndef, #else, #elif, and #endif) 9–5
9.2.1 The defined Operator 9–7

9.3 File Inclusion (#include) 9–7
9.3.1 Inclusion Using Angle Brackets (<>) 9–7
9.3.2 Inclusion Using Quotation Marks (" ") 9–8
9.3.3 Macro Substitution in #include Directives 9–8

9.4 Specifying Line Numbers (#line and #) 9–8

9.5 Implementation-Specific Preprocessor Directive (#pragma) 9–9
9.5.1 #pragma [no]builtins Directive 9–9
9.5.2 #pragma [no]inline Directive 9–9
9.5.2.1 Restrictions on Inline Expansion 9–10
9.5.3 #pragma [no]member_alignment Directive 9–11

9.5.4 #pragma [no]standard Directive 9–11

Chapter 10 Predefined Macros and Built-In Functions

10.1 Predefined Macros 10–1
10.1.1 System-Identification Macros 10–1
10.1.2 CC$gfloat (G_Floating Identification Macro) 10–2
10.1.3 The __DATE__ Macro 10–2
10.1.4 The __FILE__ Macro 10–2
10.1.5 The __LINE__ Macro 10–3
10.1.6 The __TIME__ Macro 10–3

10.2 Built-In Functions 10–3
10.2.1 Add Aligned Word Interlocked (_ADAWI) 10–4
10.2.2 Branch on Bit Clear-Clear Interlocked (_BBCCI) 10–4
10.2.3 Branch on Bit Set-Set Interlocked (_BBSSI) 10–4
10.2.4 Find First Clear Bit (_FFC) 10–5
10.2.5 Find First Set Bit (_FFS) 10–5
10.2.6 Halt (_HALT) 10–6
10.2.7 Insert Entry into Queue at Head Interlocked (_INSQHI) 10–6
10.2.8 Insert Entry into Queue at Tail Interlocked (_INSQTI) 10–6
10.2.9 Insert Entry in Queue (_INSQUE) 10–7
10.2.10 Load Process Context (_LDPCTX) 10–7
10.2.11 Locate Character (_LOCC) 10–7
10.2.12 Move from Processor Register (_MFPR) 10–8
10.2.13 Move Character 3 Operand (_MOVC3) 10–8
10.2.14 Move Character 5 Operand (_MOVC5) 10–9
10.2.15 Move from Processor Status Longword (_MOVPSL) 10–10
10.2.16 Move to Processor Register (_MTPR) 10–10
10.2.17 Probe Read Accessibility (_PROBER) 10–10
10.2.18 Probe Write Accessibility (_PROBEW) 10–11
10.2.19 Read General-Purpose Register (_READ_GPR) 10–11
10.2.20 Remove Entry from Queue at Head Interlocked (_REMQHI) 10–11
10.2.21 Remove Entry from Queue at Tail Interlocked (_REMQTI) 10–12
10.2.22 Remove Entry from Queue (_REMQUE) 10–12
10.2.23 Scan Characters (_SCANC) 10–13
10.2.24 Simple Read (_SIMPLE_READ) 10–13
10.2.25 Simple Write (_SIMPLE_WRITE) 10–14
10.2.26 Skip Character (_SKPC) 10–14
10.2.27 Span Characters (_SPANC) 10–15
10.2.28 Save Process Context (_SVPCTX) 10–15
10.2.29 Write General-Purpose Register (_WRITE_GPR) 10–15

Appendix A The lk Linker

A.1 The lk Command Line A–1

A.2 Linker Processing A–3
A.2.1 Program Section Attributes A–4
A.2.2 Virtual Memory Allocation by the Linker A–5
A.2.3 Special Processing for Modules Produced by the ld Linker A–6

Appendix B Diagnostic Messages

B.1 Diagnostic Messages from the vcc Command B–1

B.2 Diagnostic Messages from the VAX C Compiler B–2

B.3 Diagnostic Messages from the lk Linker B–33

Appendix C Transporting VAX C Programs Between VMS and ULTRIX Systems

C.1 Transporting VAX C Programs and Other ASCII Files C–1
C.1.1 Using DECnet-ULTRIX to Copy ASCII Programs C–1
C.1.2 Using DECnet–VAX to Copy ASCII Programs C–2
C.1.3 DEC/Shell on a VMS System — The tar Utility C–2

C.2 Compiling and Linking Considerations C–3
C.2.1 Input and Output Files C–3
C.2.2 Search Paths Used by the vcc Command C–4
C.2.3 Psect Differences C–4
C.2.4 Image Size Differences C–4

C.3 Transferring Data Files Between VMS and ULTRIX Systems C–5

Appendix D Language Summary

D.1 The vcc Command D–1

D.2 Data-Type Keywords D–2

D.3 Precedence of Operators D–4

D.4 Statements D–4

D.5 Conversion Rules D–4

D.6 VAX C Escape Sequences D–5

D.7 Preprocessor Directives D–5

Index

Examples
2–1 Sample Listing of Source Code 2–19
2–2 Sample Listing of Machine Code 2–20
2–3 Sample Storage Map Section 2–21
2–4 Object Module Synopsis 2–24
2–5 Program Section Synopsis 2–25
2–6 Symbol Cross Reference 2–27
2–7 Symbol Value Listing 2–28
2–8 Image Synopsis 2–29

2–9 Link Run Statistics Synopsis 2–30
3–1 Sample VAX C Program 3–15
3–2 Sample Debugging Session 3–16
4–1 Simple Addition in VAX C 4–3
4–2 Output of Information 4–5
4–3 Output Using the Newline Character 4–6
4–4 Conditional Execution Using the **if** Statement 4–7
4–5 Conditional Execution Using the **switch** Statement 4–8
4–6 Looping Using the **do** Statement 4–9
4–7 Looping Using the **for** Statement 4–11
4–8 Character String Constants and Arrays 4–15
4–9 Single Storage Allocation of Unions 4–16
4–10 Structures 4–17
4–11 Case Conversion Program 4–18
4–12 Declaring Functions 4–23
4–13 Declaring Functions Passed as Arguments 4–27
4–14 Echo Program Using Command-Line Arguments 4–29
4–15 Scope of Variable Declarations in Nested Blocks 4–32
5–1 Using the switch Statement to Count Blanks, Tabs, and Newlines 5–5
7–1 The Rules for Initializing Structures 7–19
8–1 Scope and Externally Defined Variables 8–5
8–2 Reinitializing auto Variables 8–8
8–3 Using Global Variables 8–12
8–4 Using the **globalvalue** Specifier 8–14
9–1 Nested Substitution Directives 9–2

Figures

2–1 Commands for VAX C Program Development 2–2
4–1 rvalues, lvalues, and Assigning Pointers 4–13
4–2 The Indirection Operator in Assignments 4–14
6–1 Boolean Algebra and the Bitwise Operators 6–12

Tables

2–1 Options Supported by the vcc Command 2–8
2–2 Arguments to the –V Option of the vcc Command 2–10
3–1 dbx Operators 3–4
3–2 dbx Command Summary 3–5
4–1 VAX C Keywords 4–30
6–1 VAX C Operators 6–5
6–2 Precedence of VAX C Operators 6–6
7–1 VAX C Data-Type Keywords 7–2
7–2 The Size and Range of VAX C Integers 7–3
7–3 VAX C Escape Sequences 7–6
7–4 The Size and Range of C Floating-Point Numbers 7–6
7–5 Legal C Declarations 7–25
7–6 Illegal Declarations 7–26
8–1 VAX C Storage Classes and Storage-Class Specifiers 8–3

8–2 Scope and the Storage-Class Specifiers 8–4
8–3 Location, Lifetime, and the Storage-Class Keywords 8–7
A–1 Command Options Supported by the lk Linker A–2
B–1 Linker Diagnostic Messages B–33

Preface

This guide provides reference information for using VAX C on ULTRIX ™ systems. It also contains information on how to develop and debug VAX C programs on the ULTRIX operating system running on VAX hardware. VAX C is not intended for use on RISC hardware.

Intended Audience

This guide is intended for experienced and novice programmers who need reference information on VAX C for ULTRIX systems.

Document Structure

This guide has ten chapters and four appendixes as follows:

Chapter 1 describes portability considerations for migrating C source programs between different compilers and the VMS and ULTRIX operating systems.

Chapter 2 explains how to create and compile and link VAX C programs. It also describes the forms of compiler output that you can select.

Chapter 3 discusses the debugging facilities provided by the dbx debugger and how to use the dbx commands.

Chapter 4 explains the structure of VAX C programs, including an introduction to the language, methods of controlling program flow, and the fundamental structures such as function definitions, keywords, blocks, and comments.

Chapter 5 describes the VAX C statements that provide flow control, conditional executions, looping, and interruption.

Chapter 6 discusses the expressions and operators available in VAX C, including unary, binary, conditional, comma, and assignment. Chapter 6 also explains the rules for data-type conversions.

Chapter 7 explains the data types and declarations that VAX C supports.

Chapter 8 describes the storage classes and allocation.

Chapter 9 explains the purposes and appropriate uses of the various VAX C preprocessor directives.

Chapter 10 explains the purposes and appropriate uses of the various VAX C predefined macros and builtin functions.

™ ULTRIX is a trademark of Digital Equipment Corporation.

Appendix A describes how to use the lk linker as a separate tool for linking, instead of using the **vcc** command, which both compiles and links programs.

Appendix B lists all the diagnostic messages produced by the **vcc** command program and the VAX C compiler.

Appendix C describes the mechanisms available to assist in transporting C programs between the VMS and ULTRIX operating systems.

Appendix D provides a summary of the **vcc** command and the language elements of VAX C.

Associated Documents

You may find the following documents useful when programming in VAX C. The last two documents are included if you want to transport VAX C programs between the ULTRIX and VMS operating systems.

- *The C Programming Language*[1] — Provides a more intensive tutorial than that found in the beginning of Chapter 4 of this guide.

  VAX C contains additional features and enhancements to the C language as it is defined in *The C Programming Language*. Therefore, use this guide as the reference for a full description of VAX C.

- *ULTRIX Documentation Set* — Provides information about the ULTRIX operating system and its utilities.

- *Guide to VAX C* — Provides tutorial information that describes using VAX C on the VMS operating system.

- *VMS Master Index* — Provides information on the VAX machine architecture in the VMS operating system environment. (This index identifies manuals that cover individual topics about using the VMS operating system.)

Conventions

Convention	Meaning
RETURN	The symbol RETURN represents a single stroke of the RETURN key on a terminal.
CTRL/X	The symbol CTRL/X, where letter X represents a terminal control character, is generated by holding down the CTRL key while pressing the key of the specified terminal character.
% cprog RETURN	In interactive examples, the user's response to a prompt is printed in red; system prompts are printed in black.

[1] Brian W. Kernighan and Dennis M. Ritchie, *The C Programming Language* (Englewood Cliffs, New Jersey: Prentice Hall, 1988).

Convention	Meaning
float x; . . . x = 5;	A vertical ellipsis indicates that not all of the text of a program or program output is shown. Only relevant material is shown in the example.
option, . . .	A horizontal ellipsis indicates that additional parameters, options, or values can be entered. A comma that precedes the ellipsis indicates that successive items must be separated by commas.
[output-source, . . .]	Square brackets, in function synopses and a few other contexts, indicate that a syntactic element is optional. Square brackets are not optional, however, when used to delimit the dimensions of a multidimensional array in VAX C source code.
sc-specifier ::= **auto** **static** **[extern]** **register**	In syntax definitions, items appearing separate lines are mutually exclusive alternatives.
[a | b]	Braces surrounding two or more items separated by a vertical bar (|) indicate a choice; you must choose one of the two syntactic elements.
Δ	A delta symbol is used in some contexts to indicate a single ASCII space character.
auto storage class **fprintf** function	Boldface type identifies language keywords and the names of independently compiled external functions.